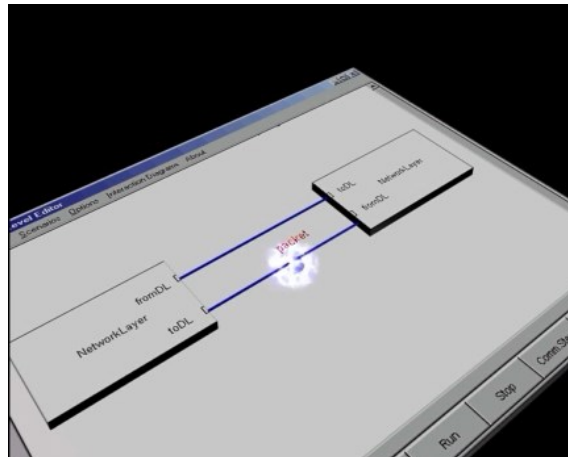


SHESim

Starter Guide



Marc Geilen, December 2000

Revisions:

Simona Vlad, December 2004

Marc Geilen, March 2005

Marc Geilen, March 2013

Electronic Systems Group
Faculty of Electrical Engineering
Eindhoven University of Technology

Introduction

This document will guide you through a first session with the SHESim tool. The SHESim tool is a simulator for the POOSL (Parallel Object-Orient Specification Language) language, part of the specification and design methodology SHE (Software/Hardware Engineering).

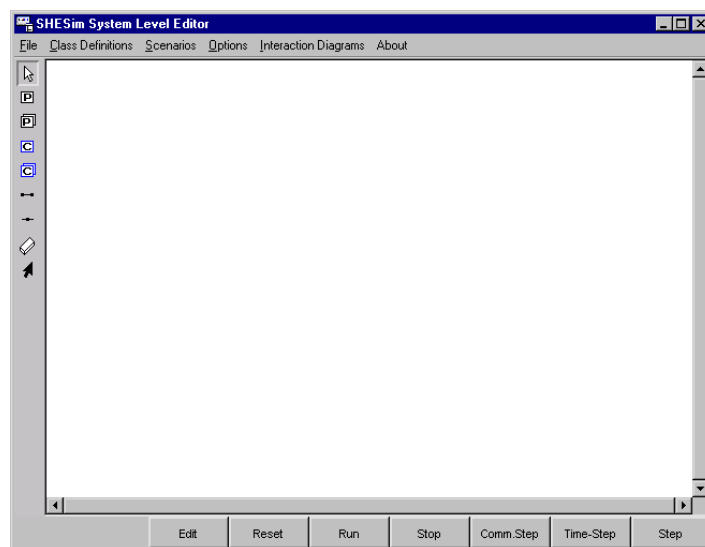
POOSL is a formal language with a mathematically defined semantics. It supports the specification of concurrent objects, composed in a static structure of communicating components in an object-oriented fashion.

What do you need?

In order to go through this guide you need to have the SHESim tool installed and you need the example called 'par' (the Positive Acknowledgement with Retransmission protocol). Both the SHESim tool and the example can be downloaded from the POOSL homepage at <http://www.es.ele.tue.nl/poosl>.

Getting Started

Structure and behaviour of POOSL models are specified by different kinds of classes. From these classes instances can be created which exhibit this behaviour. The tool then allows one to simulate this behaviour interactively. In this document we describe a first session of using the tool on an example model. More detailed information about the tool can be found in the user manual.



The System Level Editor

It is assumed that you have installed SHESim according to the instructions. When you **start SHESim**, you will see a blank window, where a graphical representation of the POOSL specification will appear. On the left there is a tool bar with different tools for manipulating the specification. At the top there is a menu bar and at the bottom there are buttons used during simulation. Furthermore you will often use the right mouse button on objects to access their specific operations.

There are some peculiarities with the VisualWorks environment in which the tool was programmed. **Click the right mouse button in the middle of the window and check if you see the following menu:** 'Process Classes / Cluster Classes / Data Classes' if you do you can skip the following paragraph, otherwise you need to follow the instructions.

Operating the right mouse button

If the right mouse button does not operate as desired, **select from the menu bar:** 'Options -> Swap Middle and Right Mouse Button'. Now the right mouse button should operate correctly.

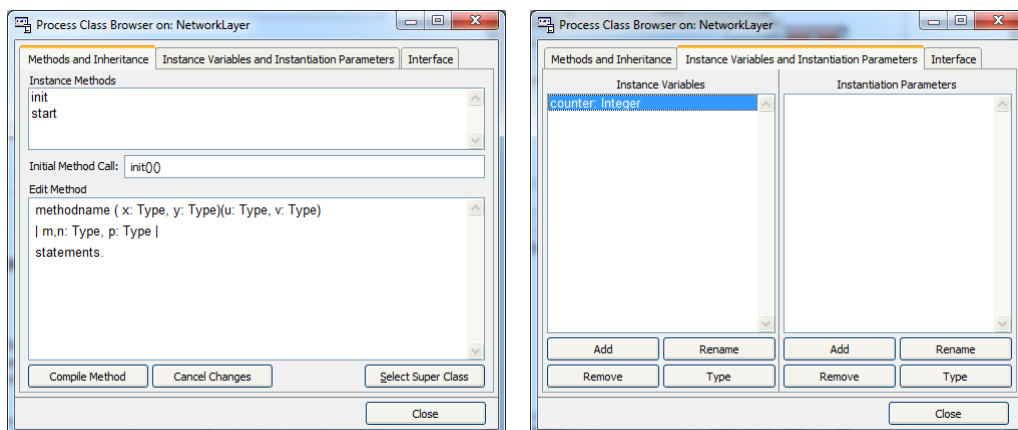
Opening a model

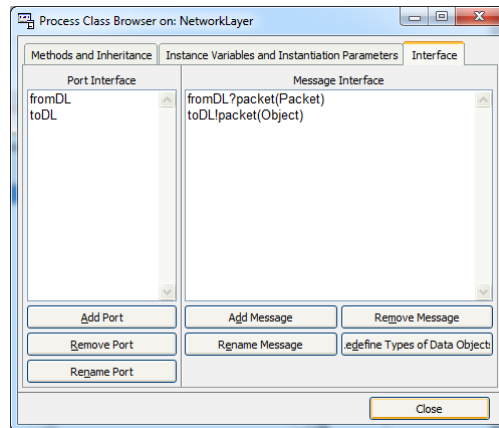
There are several ways to open and save models or classes. There are five kinds of files:

- *Files with data classes*, (ending in .xml), xml based files containing one or more definitions of data classes.
- *Files with process classes*, (ending in .xml), xml based files containing one or more definitions of process classes.
- *Files with cluster classes*, (ending in .xml), xml based files containing cluster classes and their graphical representations.
- System model *files* (ending in .xml), are xml based files containing the system-level model including all data, process and cluster classes.
- *Work status files* (ending in .im) are a completely different way to store information. In a work status file the current state of the SHESim tool is stored, all specifications, windows, simulations etc. are saved and can be reopened from this file. Note that work status files are stored in binary format and are very large (several megabytes).

Press the right mouse button in the middle of the window and select 'Load Model' (this is the same as selecting 'Load Model' from the 'File' menu at the top). The tool now asks us to provide the name of an xml file to load.

We assume that you have the example called 'par'. The example contains the par model in an old file format (.psl and .top files) and in the newer xml format. Provide the tool through the open file dialog with the location of the `par.xml` file and it will open the corresponding model. After compilation you will see the model appear in the window. You may have to enlarge the window a little to see the entire model. The window shows a number of black boxes (*process objects*), blue boxes (*clusters*) and *channels* between them. The boxes are instances of the corresponding classes that define their behaviour. We can view these definitions by clicking on an object with the right mouse button and selecting 'Edit class'. Click the right mouse button on the top-left object called `NetworkLayer1` and select 'Edit Class'. A new window opens showing the class definition of the process class called `NetworkLayer`.





A Process Class Browser

The Process Class Browser

Instance Methods

This window has three tabs. The first tab labelled 'Methods and Inheritance' shows the *instance methods* of the process class. When selected, the entire method is shown in the pane at the bottom of the window. Methods define the autonomous behaviour of process objects. Methods of process objects are not executed in automatic response to the reception of a message, such as for instance in objects of C++. A process initiates its own methods and can use explicit send and receive statements to communicate with other objects. Click on the method called `start`. It will appear at the bottom pane. You can see the method `start() ()`. The two pairs of empty braces show that the method has no input parameters and no return parameters. '| p: Packet |' indicates that there is a local variable 'p' of type 'Packet'. The body of the method consists of a select statement followed by a (tail recursive) method call of the method itself. The select statement (`sel ... or ... les`) allows the choice between two alternative behaviours. The first is sending (!) of a message called `packet` on the port named `toDL` with as a parameter the object that is the result of evaluating the expression '`new(Packet) setValueTo(counter)`'. After this the value of `counter` is increased by one. The second alternative is the reception (?) of a message called `packet` on port `fromDL` with a parameter that will be bound to the variable 'p'.

After editing a method click 'Compile Method' to compile and store it and to update the port and message interfaces, you can click 'Cancel' to return to the version before the editing. A new method can be added by typing it in the bottom pane and choosing compile. If no method is selected in the instance methods pane, the bottom pane shows a template for the creation of a new method.

Initial Method Call

When an object of the particular class is created, it begins executing the initial method call. This is specified in the single line above the 'Edit Method' pane. It contains a method call statement and initiates the behaviour of the object.

The Super Class

Furthermore there is a button labelled 'Select Super Class' at the bottom right of the window. When this button is clicked one is prompted to designate another class as the super class of this class. If a process class has a super class, it inherits the instance variables, instantiation parameters and methods of the super class. The `NetworkLayer` class has no super class.

Instance Variables

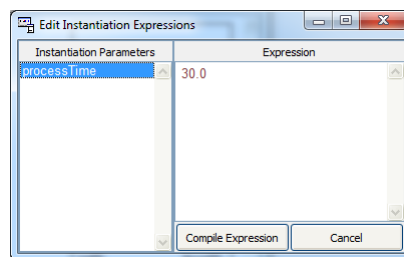
The second tab shows the *instance variables* of the class. Every process object has instance variables that can be used to hold on to data objects. The instance variables of a process object can only be accessed by the object itself. Sharing of data between process objects is not possible.

The `NetworkLayer` class has only one instance variable named `counter` of type `Integer`. To add, remove or rename instance variables or change the type of an instance variable, use the buttons at the

bottom of the window, or operate the right mouse button on the instance variables pane or on a selected instance variable and select the appropriate option. Note that in POOSL the type of a variable is for documentation purposes only. At runtime a variable can be bound to any kind of object, no matter what the type of the variable or the class of the data object it is bound to.

Instantiation Parameters

The right pane shows the so-called *instantiation parameters*. Instantiation parameters are just like instance variables. The difference is however that the value of these variables can be given an initial value for every instance of the class. Thus, this allows one to differentiate the behaviour of individual instances of the class. It allows for example to give every instance a unique identification. Our `NetworkLayer` class does not have instantiation parameters. Also here, use the buttons or operate the right mouse button to add, remove, rename or change the types of the instantiation parameters. **Switch to the System Level Editor window and click the right mouse button on the process object `DataLinkSender3`, you will see an option called 'Instantiation Parameters'. If you click it, a new window will appear showing the object's instantiation parameters. If you select a parameter on the left you can assign it a value by typing it in the pane at the right. Close the window and return to the process class browser.**



Editing Instantiation Parameters

Port Interface

The last tab of the process class browser shows on the left the *port interface* of the process class. Every process object has ports through which it can send and receive messages in order to communicate with other process objects.

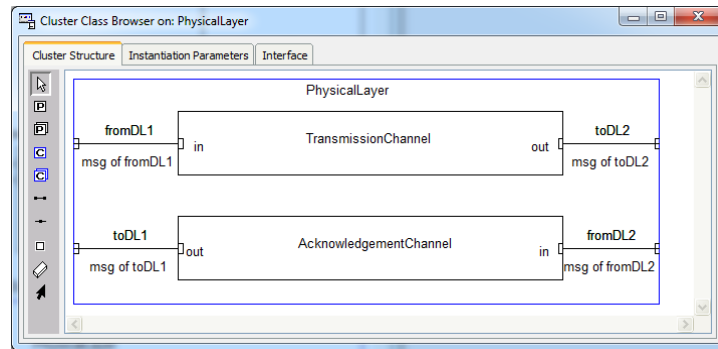
The `NetworkLayer` class has only two ports: `toDL` and `fromDL`. To add, remove or rename the ports, use the buttons at the bottom, or operate the right mouse button on the port interface pane or on a selected port and select the appropriate option. Note that the list also contains ports that are defined in the super classes of the class that is being edited. Removing and renaming of a port is allowed only if it does not belong to a super class and if the port is not used in an instance method of the edited process class or in its subclasses.

Message Interface

The right pane shows the *message interface* of the process class. Process objects communicate by sending and receiving messages. The message signatures of the messages that can be sent or received by a process object are stored in its message interface.

The `NetworkLayer` process class has two message signatures: `toDL!packet(Packet)` and `fromDL?packet(Packet)`. To add, remove, rename the message signatures or change the type of data objects of the message signatures, use the buttons at the bottom or operate the right mouse button on the message interface pane or on a selected message signature and select the appropriate option. Note that the list also contains message signatures that are defined in the super classes of the class that is being edited. Removing and renaming of a message signature is allowed only if it not belongs to a super class and if the message is not used in an instance method of the edited process class or in its subclasses.

We can now close the process class browser by clicking the close button of the window.



A Cluster Class Browser

The Cluster Class Browser

We will now take a look at the definition of cluster classes.

Click the right mouse button on the blue box at the bottom of the System Level Editor called `PhysicalLayer` and select 'Edit class'. A cluster class is similar to the specification in the System Level Editor and has instantiation parameters like process classes.

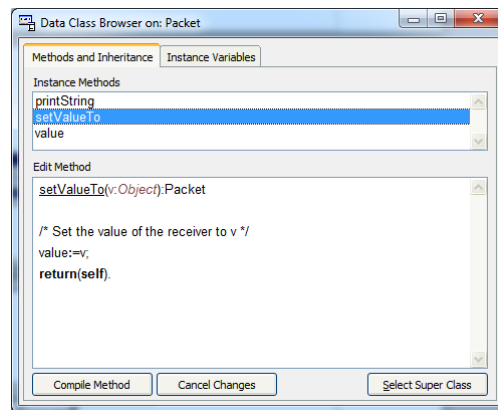
The structure of a cluster class is created by placing process objects and other clusters inside it and connecting them via channels using the appropriate tools from the toolbar on the left (see the manual). To allow a cluster to communicate with the outside world one can add ports. A cluster of class `PhysicalLayer` has four ports called `fromDL1`, `toDL1`, `fromDL2` and `toDL2`. These ports are connected via channels called with identical names to the two process objects '`TransmissionChannel`' and '`AcknowledgementChannel`' both of class '`SimplexChannel`'. **Inspect the instantiation parameters of '`TransmissionChannel`'** (clicking the right mouse button on the process and selecting 'Instantiation parameters'). It has the parameters: '`capacity`' set to 4, '`errorProbability`' set to 0.5 and '`transmission-Distribution`' initialised with the expression '`new(Normal) withParams(100.0,0.0)`' which yields a data object representing a source of normally distributed random variables with mean 100 and variance 0. **Close the instantiation parameter window again.**

Cluster classes also have instantiation parameters. They can be added in the tab 'Instantiation Parameters' just as it was done in process classes. The parameters of a cluster class can be used to define the parameters of processes and other clusters inside the cluster. The third tab defines the interface, similar to the process classes.

Close the cluster class browser window.

Data Classes

Click the right mouse button on a free spot in the System Level Editor window and select 'Data Classes' (note that one can also select 'Process Classes' or 'Cluster classes' here) This gives a window with a list of all data classes defined. Again using the right mouse button in this menu we can access options to add, remove rename data classes. Moreover we can load and save data classes. 'Save Model' for instance produces an xml file with the selected classes. If a data class is selected (**select the 'Packet' class**) and we **choose 'Edit'** (using the right mouse button), a new window opens similar to the process class browser window. Data classes have instance variables methods and super classes just as process classes, but they do not have instantiation parameters nor an initial method call. Data classes are very similar to objects in C++ or Smalltalk. They execute methods as a response to the reception of a message and after completion return a result to the sender of that message.



A Data Class Browser

They are not autonomous and thus have no initial method. They are also not static (as processes) and thus do not have instantiation parameters. The instance variables and methods panes operate just like the ones in the process class browsers. **Select the method called 'setValueTo'** We see that the definition of a data method is somewhat different than the definition of a process method. The methods do not have return parameters, every method returns a single object using the 'return' statement. After the method name and the parameters there is an indication of the type of object the method returns. Again this type indication is only an aid for the user. It is not checked (neither at compile time not at run-time) that the returned object is of the appropriate type. We see that comments can be added between '/*' and '*/' (in C-style). The body of the method contains an assignment 'value:=v' of the object passed as a parameter to the method call to the instance variable called 'value'. This statement is followed by the return statement (which must always be the last statement to be executed) returning a reference to the object executing this method itself.

Simulating the model

The Simulation Buttons

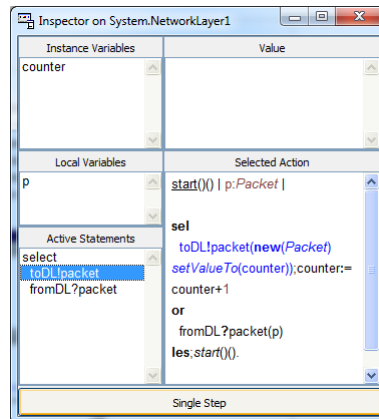
We have seen the most important aspects of the creation of a POOSL model. Once a model has been defined, it can be simulated. For this we use the buttons at the bottom of the System Level Editor window. The 'Reset', 'Run', 'Stop', 'Comm. Step', 'Time Step' and 'Step' buttons control the simulation and bring the tool in simulation mode. The 'Edit' button returns the tool to edit mode. When in simulation mode, the model cannot be edited. To understand the simulation buttons we need to know a bit about the way processes behave. Processes can execute atomic actions, taking no time or make time steps. Actions of processes are asynchronous and interleaved, i.e. actions of different processes are never made together (unless it involves a synchronisation between the two processes). Time steps on the other hand are made synchronously, by all the processes at the same time. Furthermore actions of processes have priority of time transitions. Time will not advance as long as there are process objects that can execute actions immediately.

Now the 'Reset' button initialises the objects in the model. The 'Step' button lets the model execute a single step, a single action of one of the processes, a synchronisation between two processes or a single time transition of all processes. After the step, simulation stops. The 'Time-Step' button start simulation and stops after the first time transition has been made. The 'Comm-Step' button starts simulation and stops after a communication has occurred on one of the channels that are currently visible. A communication inside a cluster that is not currently visible will not halt the simulation. 'Run' will start simulation without halting. A running simulation can be terminated by the 'Stop' button.

If a communication takes place on a channel, the channel will light up in red and the name of the message will be visible besides the channel. **Press the 'Run' button.** The simulation will start and you will see communications taking place on the channels. **'Press the 'Stop' button'** and the simulation will halt. At the bottom left corner of the window we can see the current simulation time. You can experiment with the simulation buttons and see how the simulator responds.

Inspecting Objects

To gain a better understanding of what is happening inside the model, it is possible to inspect objects. This is done by pressing the right mouse button on an object and selecting 'Inspect' from the menu. **Open an inspector on the process object 'NetworkLayer1'.** A new window opens showing the state of the process object. On the left-hand side we can select instance variables (including instantiation parameters) and local variables. If we select a variable, its value is displayed in the upper right pane. We can also click the right mouse button on a variable and choose 'inspect' to open a new inspector window for the data object itself (unless it is a primitive object).



A Process Inspector

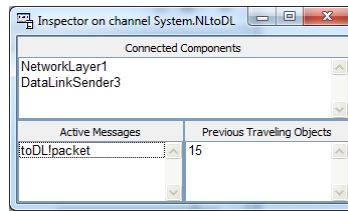
At the bottom of the inspector we see a pane labelled 'active statements'. As processes may have concurrent behaviour, several statements may be active at the same time. These statements are shown in this pane as well as the composite statements of which they are a part. If such a statement is selected, the corresponding method body is displayed in the right pane labelled 'Selected Action'. And the statement is highlighted in one of the following colours:

- blue, if it is a statement that is ready to be executed,
- green if it is a composite statement (such as a select statement or interrupt),
- red if its execution is prohibited by a guard that is false,
- purple if the test of its executability did not succeed, this signifies an error in the model.

At the bottom of the inspector there is a button labelled 'Single Step' this button can be used to start simulation and halt again after this particular process object has performed some action.

Also clusters can be inspected. **Close the inspector on 'NetworkLayer1' and open an inspector on the cluster 'PhysicalLayer'.** A new window opens showing the internal structure of the cluster. When a simulation is running also on these channels, messages will appear. The objects inside of the cluster can of course be inspected again.

Inspectors can also be opened on channels by clicking on them using the right mouse button. **Click on the channel named 'NLtoDL' in the System Level Editor window.** A new window opens. In the upper pane it shows all process objects connected to the channel, in this case 'DataLinkSender3' and 'NetworkLayer1'. In the lower left pane there is a list of active messages on this channel. If a message is listed here, then one of the connected processes is trying to synchronise on this channel using this message. At the lower right pane labelled 'Previous Travelling Objects' we see a list of objects that were passed as parameters of a message during the previous synchronisation that took place on this channel (these parameters are also called 'travelling objects').

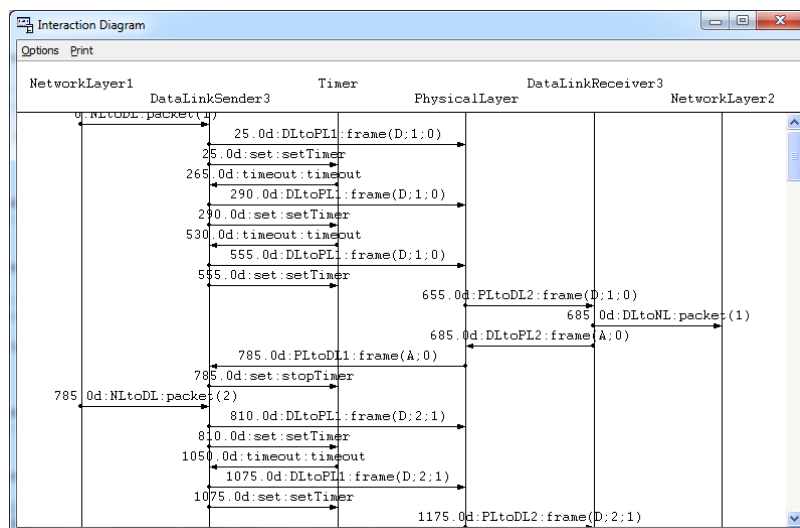


A Channel Inspector

Interaction diagrams

There is one other powerful way to view the results of the simulations by means of an interaction diagram. It can be opened by clicking the right mouse button on a free spot in the System Level Editor and selecting 'Messages' or by selecting 'Interaction Diagrams -> Open Interaction Diagram' from the menu. A new window opens showing vertically lines with the name of the objects in the simulation at the top. Simulation time is shown on the vertical axis and arrows between vertical lines show messages that have been exchanged between the processes. These arrows are labelled with a time stamp, the name of the message and the parameters that were passed.

The order of the objects can be changed by clicking and dragging their names to the left or the right. Furthermore it is possible to replace a cluster by its internal structure or dive into a cluster forgetting about the objects surrounding it. This is done by pressing the right mouse button on the name of an object and selecting one of the navigation options 'explode', 'hide', 'enter' or 'leave'.



An Interaction Diagram

Saving a Specification

We have used the 'Load Model' option and we have seen that we can load and save classes in the Data / Process or Cluster Classes windows. There are also corresponding save operations. The 'Save Model' option is the counterpart of 'Load Model' and there are the Load and Save Top Level Diagram operations, they only save the design as it is seen in the System Level Editor Window, but not the process / data and cluster classes supporting it! For a complete model, four files are required, for data classes, process classes, cluster classes and the system level diagram.

There is also another, entirely different way to save your work, by saving the work status. This can be done by selecting 'Save Work Status' from the 'File' menu. You will also be prompted if you want to save your work when you close SHESim. The work status file contains the entire program in its current state, including all class definitions and specifications as well as for instance the current state of the simulation. The disadvantage is that a work status file is very large. It is several megabytes large. These files are opened directly, not by first opening SHESim.

Other Sources of Information

There is a manual for a more elaborate description of the use of the SHESim tool and a document describing the syntax of the POOSL language. There is also presentation on the SHE methodology and the POOSL language. These documents can be obtained via internet on the locations below

Web Pages

The following web pages provide information about POOSL, and the SHESim and Rotalumis tools.

- POOSL homepage:
<http://www.es.ele.tue.nl/poosl>

E-mail

If you have comments for the improvement of this guide please let us know! Also for other questions, you can contact us via

M.C.W.Geilen@tue.nl